CAM-BASED SEARCH ENGINE DEVICES HAVING ADVANCED SEARCH AND LEARN INSTRUCTION HANDLING

Reference to Priority Application

This application is a continuation-in-part (CIP) of U.S. Application Serial No. _____, filed November 14, 2003 (Attorney Docket No. 5646-118), which is a continuation-in-part of U.S. Application Serial No. 10/698,246, filed October 31, 2003, which claims priority to U.S. Provisional Application Serial No. 60/516,178, filed October 31, 2003, the disclosures of which are hereby incorporated herein by reference.

Field of the Invention

The present invention relates to integrated circuit devices that support search operations and, more particularly, to CAM-based search engine devices and methods of operating same.

Background of the Invention

Conventional network processor units (NPU) may be interfaced to integrated IP coprocessors (IIPC) in a manner that enables both SRAMs and IIPCs to be operated on the same memory mapped bus. As illustrated by FIG. 1, a conventional IIPC **30** may be coupled through a standard memory mapped interface to an NPU **10**, which operates as a command source. The address bits ADDR[23:22] represent a two-bit select field that identifies one of four possible IIPCs on the bus for which a read operation is directed. The NPU **10** may include an SRAM controller that is based on FIFO communication. The SRAM controller includes internal bus control state machines **20** and pin control state machines **14**. Data and address information is transferred between these state machines using push and pull data FIFOs **12a** and **12d** and read and write command FIFOs **12b** and

12c that supply read and write addresses to the pin control state machines **14**.

The IIPC **30** is illustrated as including a content addressable memory (CAM) core **36** and logic **38** that couples the CAM core **36** to the memory mapped interface. This memory mapped interface is illustrated as including read control logic **32** and write control logic **34**. The write control logic **34** is configured to receive an address ADDR[21:0], a write enable signal WE_N[1:0], input data DATAIN[15:0] and input parameters PARIN[1:0]. The read control logic **32** is configured to receive the address ADDR[21:0] and a read enable signal RE_N[1:0] and generate output data DATAOUT[15:0] and output parameters PAROUT [1:0]. Like the SRAM controller within the NPU **10**, this memory mapped interface is based on FIFO communication. The IIPC **30** performs operations using the input data DATAIN[15:0] and input parameters PARIN[1:0] and then passes back result values to the NPU **10**. The timing between the receipt of the input parameters and the return of the corresponding result values is not fixed. Instead, it is determined by the amount of time the IIPC **30** requires to execute the specified instruction and depends on the number and type of other instructions currently pending within the IIPC **30**.

These pending instructions are initially logged into respective instruction control registers **50** that support a plurality of separate contexts (shown as a maximum of 128). These instructions may be processed in a pipelined manner. The result values generated at the completion of each context are provided to respective result mailboxes **40**. The validity of the result values within the mailboxes **40** is identified by the status of the done bit within each result mailbox **40**. Accordingly, if a read operation is performed before the result values are ready, the NPU **10** will be able to check the validity of the done bit associated with each set of result values to determine whether the corresponding values of valid. However, because there can be multiple contexts in progress within the IIPC **30** at any given time and because the completion of the contexts does not necessarily

occur in the same sequence as the requests were made, the NPU **10** may need to regularly poll the result mailboxes **40** at relatively high frequency to obtain new results as they become valid. Unfortunately, such regular polling can consume a substantial amount of the bandwidth of instructions that are issued to the IIPC **30** and lead to relatively high levels of operational inefficiency when the IIPC **30** is running a large number of contexts. Thus, notwithstanding the IIPC **30** of FIG. 1, which is capable of supporting a large number of contexts, there continues to be need for more efficient ways to communicate result status information from an IIPC to a command source, such as an NPU.

Referring now to FIG. 2A, another conventional IIPC **300** may include an aging feature that automatically removes stale entries from an internal CAM core **330**. This aging feature can be operated as a fully independent hardware function requiring no software intervention or as a software-managed procedure with hardware assist. The IIPC **300** of FIG. 2A includes a memory mapped interface **302** having a write interface **304** and a read interface **306** therein. These write and read interfaces **304** and **306** may be configured as quad data rate interfaces that communicate to and from a command source (e.g., ASIC or NPU) having a compatible interface. A clock generator circuit **308** may also be provided that is responsive to an external clock EXTCLK. This clock generator circuit **308** may include delay and/or phase locked loop integrated circuits that operate to synchronize internal clocks within the IIPC **300** with the external clock EXTCLK. A reset circuit **310**, which is configured to support reset and/or power-up operations, is responsive to a reset signal RST. Context sensitive logic **312** may support the processing of multiple contexts. The context sensitive logic **312** may include an instruction memory **316** that receives instructions from the write interface **304** and a results mailbox **314** that may be accessed via the read interface **306**. The instruction memory **316** may be configured as a FIFO memory device. The results mailbox

314 is a context specific location where the IIPC **300** places results returned from a previously issued command.

The internal CAM core **330** is illustrated as a ternary CAM core that contains a data array and a mask array **328**. This CAM core **330** may be configurable into a plurality of independently searchable databases. General and database configuration registers **318** are also provided along with global mask registers GMRs **320**. These registers provide data to instruction loading and execution logic **332**, which may operate as a finite state machine (FSM). The instruction loading and execution logic **332** communicates with the CAM core **330** and the result logic **334**. If the IIPC **300** is configured to support a depth-cascaded mode of operation, a cascade interface **338** may be provided for passing data and results to (and from) another IIPC (not shown). The instruction loading and execution logic **332** may also pass data to and from an external memory device, via an SRAM interface **336**.

The aging logic **321** is illustrated as including two memory arrays: an age enable array **322** and an age activity array **324**. These memory arrays may have bit positions that map directly to entries within the CAM core **330**. Thus, if the CAM core **330** has 128k entries (e.g., x72 entries), then the age enable array **322** and age activity array **324** may each have a capacity of 128k bits. The illustrated aging logic **321** may operate with the instruction loading and execution logic **332** to (i) reset age activity bits that have been previously set within the age activity array **324** in response to successful search operations and (ii) age out entries associated with previously reset activity bits by invalidating the corresponding entries.

The aging operations may include periodically inserting an aging instruction into an instruction pipeline within the IIPC **300**. As illustrated by the global and database aging request circuit **350** of FIG. 2B, a global aging register **352** (e.g., 32-bit countdown counter) may be used to specify the number of cycles of a system clock SYSCLK that are to occur before each aging operation request is inserted into the instruction pipeline. Each

aging operation that is inserted may operate to age one entry within a database that is programmed to support aging. Each database within the CAM core **330** may have an individually specified time period for aging, which means the frequency of the age service requests for the plurality of databases (shown as DB0-DB15) may be independently controlled. These time periods may be specified by a plurality of 24-bit countdown counters **356** that are set to database specific time constants (i.e., count values) and clocked at 1/256th the system clock frequency. This slower clocking rate may be achieved with a divide-by-8 circuit **354** that is responsive to the system clock SYCCLK. As long as a database is enabled for aging, a database age service request is issued every time the corresponding 24-bit countdown counter **356** decrements to zero and is reinitialized. The IIPC **300** determines which database is to be serviced during each aging operation using a round-robin arbitration of all pending database age service requests. One entry within a selected database is aged in response to a selected age service request. The aging of a selected entry proceeds as follows. If a corresponding age enable bit for the entry is set to 0 within the age enable array **322**, then the aging operation does nothing because the entry is not subject to aging. If the age enable bit is set to 1 within the age enable array **322** and a corresponding age activity bit is set to 1 (i.e., the entry is active) within the age activity array **324**, then the aging operation clears (i.e., resets) the age activity bit to 0. Finally, if the age enable bit is set to 1 within the age enable array **322** and the corresponding age activity bit is set to 0 (i.e., the entry is inactive), then the aging operation removes the entry from the selected database by marking the entry as invalid (e.g., sets the valid bit associated with the entry in the CAM core **330** to an invalid state). The activity bit associated with an entry can be set to 1 whenever the entry is originally written into the CAM core **330** or a search operation results in a hit for the corresponding entry. A learn instruction and a set valid instruction may also operate to set an activity bit associated with a corresponding entry.

Conventional CAM-based search engine devices may also be configured to perform search and learn (SNL) operations that include an initial search operation and, if necessary, a subsequent learn operation of a new entry into a CAM core. As illustrated by FIG. 5 of U.S. Patent No. 6,219,748, operations may be performed to decode a learn instruction and load comparand data (e.g., search key) into a comparand register within a search engine device. An operation is then performed to search a CAM array using the comparand data as a search word. If a match occurs in response to the search operation, then operations associated with generating conventional search results are performed. However, if no match is present and the CAM array is not full, then the comparand may be written as a CAM entry into an internally generated next free address within the CAM array. This write operation into a next free address is treated as a learn operation as opposed to a conventional write operation that includes an externally supplied write address. The next free address associated with the newly written CAM entry is then reported to a command host. Related CAM-based search engine devices that describe learn operations are also disclosed in U.S. Patent Nos. 6,148,364 and 6,240,485.

Summary of the Invention

CAM-based search engine devices operate to reduce the occurrence of duplicate learned entries within a CAM database when processing search and learn (SNL) instructions. According to first embodiments of the present invention, a search engine device is configured to support processing of equivalent SNL instructions in a manner that reduces the occurrence of duplicate learned entries within a CAM database even when the equivalent SNL instructions are received by the search engine device during immediately consecutive time intervals. SNL instructions are treated as equivalent when they are associated with the same search key and directed at the same database(s) within a CAM core. In particular, a search engine device is configured to process first and second immediately consecutive and equivalent SNL instructions as:

(i) a first SNL instruction, and (ii) a second search and search (SNS) instruction, respectively. This preferred processing is performed in order to block an addition of a duplicate learned entry within a database in the search engine device. The search engine device may also be configured to support processing of the first and second immediately consecutive and equivalent SNL instructions in a manner that prevents a failure to learn a new entry when the corresponding database is full. In particular, the internal conversion of an SNL instruction into an SNS instruction can be suspended when the corresponding database is full.

According to second embodiments of the present invention, a CAM-based search engine device is provided that is configured to convert a learn portion of a search-and-learn (SNL) instruction associated with a search key into a search operation using the search key. This conversion operation is performed in response to detecting a prior equivalent learn of the search key in the search engine device. The search engine device may also include a SNL cache memory device that is configured to store search keys that accompany SNL instructions and marker information (e.g., flags) that identifies whether the stored search keys are duplicates of other search keys within the SNL cache memory device. This SNL cache memory device may operate as a first-in first-out (FIFO) memory device.

Still further embodiments of the present invention include methods of operating CAM-based search engine devices. These methods include searching a database within a CAM core with a search key to detect the presence of a hit or miss condition. Then, if a miss condition has been detected, as step is performed to check the search key to determine whether is has been marked as a duplicate. If the search key has been marked as a duplicate, then the checking step is followed by the step of converting a learn portion of the SNL instruction into a search operation that results in another search of the database with the search key.

Brief Description of the Drawings

FIG. 1 is a block diagram of a network processor unit having an SRAM controller therein that is coupled to a conventional integrated IP-coprocessor (IIPC).

FIG. 2A is a block diagram of a conventional IIPC that supports automatic aging of CAM core entries.

FIG. 2B is a block diagram of multi-bit counters that are configured to generate aging operation requests and age services requests within the IIPC of FIG. 2A.

FIG. 3 is an electrical schematic that illustrates an integrated search engine device having result status signaling, according to embodiments of the present invention.

FIG. 4 is a block diagram of an integrated circuit system that includes a pair of network processor units (NPUs) and an integrated search engine device having two quad data rate interfaces, according to embodiments of the present invention.

FIG. 5 is a block diagram of a CAM-based search engine device with per entry age reporting capability, according to embodiments of the present invention.

FIG. 6 is a flow diagram of operations that illustrates methods of reporting entries that have been aged out of a search engine device, according to embodiments of the present invention. FIG. 6 includes FIGS. 6A and FIGS. 6B.

FIG. 7A illustrates a plurality of memory devices that may be used in a aging control circuit illustrated in FIG. 5.

FIG. 7B illustrates the mapping of bit positions within an age report enable memory array to a CAM core illustrated in FIG. 5.

FIG. 8 is a block diagram that illustrates how the search engine device of FIG. 5 may be depth-cascaded in a system that supports per entry age reporting across multiple search engine devices.

FIG. 9 is a block diagram of a search engine device that is configured to block the learning of duplicate entries in response to search and learn (SNL) instructions.

FIG. 10 is a flow diagram of operations that illustrate methods of performing search and learn (SNL) instructions according to embodiments of the present invention.

FIGS. 11A-11H illustrate how equivalent SNL instructions that are received close in time are processed in the search engine devices of FIGS. 5 and 9.

FIG. 12 is a flow diagram of operations that illustrate additional methods of performing search (SEARCH) and learn (LEARN) instructions according to embodiments of the present invention.

Detailed Description of Preferred Embodiments

The present invention now will be described more fully herein with reference to the accompanying drawings, in which preferred embodiments of the invention are shown. This invention may, however, be embodied in many different forms and should not be construed as being limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Like reference numerals refer to like elements throughout and signal lines and signals thereon may be referred to by the same reference characters. Signals may also be synchronized and/or undergo minor boolean operations (e.g., inversion) without being considered different signals. Moreover, when a device or element is stated as being responsive to a signal(s), it may be directly responsive to the signal(s) or indirectly responsive to the signal(s) (e.g., responsive to another signal(s) that is derived from the signal(s)).

Referring now to FIG. 3, an integrated IP coprocessor (IIPC) **100** that is configured to operate as an integrated search engine device according to embodiments of the present invention will be described. This IIPC **100** includes a CAM core **120** having at least one database of

searchable entries therein. In typical embodiments, the CAM core **120** may have as many as sixteen independently searchable databases. Programmable power management circuitry (not shown) may also be integrated with the CAM core **120** so that only a selected database(s) consumes power during a search operation. CAM cores having a fewer or larger number of databases are also possible. The CAM core **120** is electrically coupled to a control circuit. The control circuit is illustrated as including a scheduler, a finite state machine and logic **110** that can support multiple overlapping contexts. The control circuit is further illustrated as including: a plurality of result mailboxes **90**, a result status register(s) **80**, a result status select register **70**, an interrupt indication circuit **60a** and a non-interrupt indication circuit **60b**. The result status register **80**, result status select register, interrupt indication circuit **60a** and non-interrupt indication circuit **60b** collective define a result status notification circuit. The result mailboxes are illustrated as having a capacity to support result values from as many as 128 contexts. These mailboxes **90** also retain information that identifies whether the result values are valid or not. Result values are valid when the respective context is complete and the result values generated by the completed context have been loaded into a respective mailbox **90**. When this occurs, the done status bit (DONE) associated with a respective mailbox **90** is set and remains set until such time as the respective mailbox **90** is read, at which point it is reset. The result status register(s) **80** is configured to retain a copy of the done status bits for the result mailboxes **90**. In the illustrated embodiment, the result status register **80** is illustrated as a 128-bit register. This register may be partitioned at 32-bit segments (i.e., four registers), which support efficient reading of the contents of the result status register **80** across a 32-bit wide bus at a single data rate (SDR) or a 16-bit wide bus at a dual data rate (DDR). The result status register **80** receives and generates a 128-bit result status signal RS<0:127>, which indicates the states of completion of a corresponding plurality of contexts being handled by the search engine device. For

example, if the result status signal RS<0:127> is set to the value of <0101000...000110>, then contexts 1, 3, 125 and 126 are done and the result values for those contexts are valid and the remaining contexts are not done.

The result status select register(s) **70** is a 128-bit programmable register that generates a result status select signal RSS<0:127>. This signal operates to select one of two indication circuits for receipt of active bits within the result status signal RS<0:127>. These indication circuits are illustrated as an interrupt indication circuit **60a** and a non-interrupt indication circuit **60b**. The interrupt indication circuit **60a** includes an interrupt generator **64** that generates an interrupt INT to the command host **140** via the memory mapped interface **130**. The interrupt generator **64** may also generate interrupts in response to other activity within the control circuit, according to a predefined protocol. In contrast, the non-interrupt indication circuit **60b** generates an asynchronous aggregate result status signal (ARS) to the command host **140** via the memory mapped interface **130**. This ARS signal is configured to have a leading edge that occurs when a first one of a selected plurality of contexts is completed and an active level that is held so long as at least one of the selected plurality of contexts remains completed (i.e., done status bit remains set).

The interrupt indication circuit **60a** has a first bank **62a** of AND gates that output to an OR gate **68a**. The non-interrupt indication circuit **60b** has a second bank **62b** of AND gates that output to an OR gate **68b**. When one or more bits of the result status select signal RSS<0:127> are set high to logic 1 levels, then the corresponding result status signals RS<0:127> are passed to the inputs of the OR gate **68a**. If any of these result status signals are switched to active logic 1 values, then the output of the OR gate **68a** will switch and cause the interrupt generator **64** to produce an interrupt INT at the memory mapped interface **130**. But, when one or more bits of the result status select signal RSS<0:127> are set low to logic 0 levels, then the corresponding result status signals RS<0:127> are passed to the

input of the OR gate **68b**. Accordingly, if the result status select signal RSS<0:127> is set so that RSS<0:127> = <00000....0000>, then the aggregate result status signal at the output of the OR gate **68b** will be switched high (or held high) whenever any of the result status bits RS<0:127> is set high to indicate the completed state of a respective context. Alternatively, if the result status select signal RSS<0:127> is set so that RSS<0:127> = <11111....1111>, then the signal at the output of the OR gate **68a** will be switched high (or held high) whenever any of the result status bits RS<0:127> is set high to indicate the completed state of a respective context. In this manner, the result status select register **70** provides programmable control over how the result status signals are to be reported to the command host **140**.

Based on the above-described configuration of the control circuit, the completion of any context within the IIPC **100** will result in the transfer of result values from the scheduler, state machine and logic **110** to a corresponding result mailbox **90**. Assuming this context represents a first-to-finish operation (e.g., lookup within the CAM core), then the setting of the respective done bit within the result mailbox **90** will result in the latching of this done information by the result status register(s) **80**. If this done information relates to context 0, then the result status signal RS<0:127> will equal <10000...000>. If the result status select register is set so that the result status select signal RSS<0:127> equals <0XXXXXX...X>, where X represents a "don't care" for purposes of this example, then the aggregate result status signal ARS will be set to an active high level and passed from the memory mapped interface **130** to the command host **140**. Alternatively, if the result status select register is set so that the result status select signal RSS<0:127> equals <1XXXXXX...X>, then the output of the OR gate **68a** within the interrupt indication circuit **60a** will switch high. This active high signal at an input of the interrupt generator **64** will result in the generation of an interrupt that passes to the memory mapped interface **130** and the command host **140**.

In response to the generation of an interrupt INT or an active high aggregate result status signal ARS, the command host **140** may issue an operation to read the result status register **80**. This operation includes generating an address ADDR[23:0] to the memory mapped interface **130**. The fields of this address are illustrated by TABLE 1. The two most significant bits of the address operate to select the particular IIPC **100** for which the read operation is destined. The seven address bits ADDR<21:15> identify a particular context within a range of 128 possible contexts. The eleven address bits ADDR<4:14> are not used. The address bit ADDR<3> represents a result status identifier (RES_STATUS). If this bit is set to a first logic value (e.g., 0), then an entry within the result mailbox **90** associated with the designated context is to be read back to the command host **140**. On the other hand, if the result status identifier is set to a second logic value (e.g., 1), then a designated portion of the result status register **80**, which identifies the value of 32 result status signals, is to be read back to the command host. The final 3-bit portion of the address, shown as ADDR<2:0>, identifies an entry value. As illustrated by TABLE 2, this entry value identifies one of eight entries to be read from the designated result mailbox **90** when the result status identifier RES_STATUS is set to a logic 0 value. Alternatively, the entry value identifies one of four portions of the result status register **80** to read from when the result status identifier is set to a logic 1 value. In this manner, four consecutive read operations may be performed to enable the command host to read the entire contents of the result status register **80** and thereby readily identify which ones of the 128 result mailboxes **90** contain valid result values.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SELECT		CONTEXT							NOT USED											RES STATUS	ENTRY VALUE		

TABLE 1

RES_STATUS	ENTRY VALUE	ACTION
0	000	READ ENTRY 0 IN CONTEXT SPECIFIC MAILBOX
0	001	READ ENTRY 1 IN CONTEXT SPECIFIC MAILBOX
0	010	READ ENTRY 2 IN CONTEXT SPECIFIC MAILBOX
0	011	READ ENTRY 3 IN CONTEXT SPECIFIC MAILBOX
0	100	READ ENTRY 4 IN CONTEXT SPECIFIC MAILBOX
0	101	READ ENTRY 5 IN CONTEXT SPECIFIC MAILBOX
0	110	READ ENTRY 6 IN CONTEXT SPECIFIC MAILBOX
0	111	READ ENTRY 7 IN CONTEXT SPECIFIC MAILBOX
1	000	READ RESULT STATUS BITS [31:0]
1	001	READ RESULT STATUS BITS [63:32]
1	010	READ RESULT STATUS BITS [95:64]
1	011	READ RESULT STATUS BITS [127:96]
1	100	RESERVED
1	101	RESERVED
1	110	RESERVED
1	111	RESERVED

TABLE 2

Referring now to FIG. 4, an integrated circuit system **200** according to another embodiment of the present invention will be described. This system **200** is illustrated as including an IIPC **100'** that is configured in accordance with the IIPC **100** of FIG. 3. In addition, the IIPC **100'** includes a pair of memory mapped interfaces **130a** and **130b** that communicate with a pair of network processor units (NPUs) **140a** and **140b**. Each memory mapped interface **130a** and **130b** is associated with respective mailboxes (**90a** and **90b**), result status notification circuits (**66a** and **66b**) and pipelined instruction circuits **112a** and **112b**. These pipelined instruction circuits **112a** and **112b** share access to a round robin scheduler and finite state machine **110a**. Logic circuits, in the form of SRAM logic **110c** and result logic **110b**, communicate with the CAM core **120** and the state machine **110a**.

Referring now to FIG. 5, a CAM-based search engine device **500** according to another embodiment of the present invention has the capability of performing age reporting on a per entry basis to a command host(s). The search engine device **500** is illustrated as including a ternary CAM core **522** and a number of surrounding logic circuits, registers and memory devices that collectively operate as a control circuit that is coupled to the CAM core **522**. This control circuit is configured to perform the functions and operations described herein. The search engine device **500** may include a peripheral controller interconnect (PCI) interface **502**, which is configured to enable a control plane processor to have direct access to the search engine device **500**. Instructions received at the PCI interface **502** are passed to an interface logic circuit **508** having an instruction memory (e.g., FIFO) and results mailbox therein. The search engine device **500** also includes a dual memory mapped interface, which is typically a dual quad data rate interface. The first memory mapped interface **504** contains a write interface and a read interface that can support communication with a network processor unit (e.g., NPU 0). The second memory mapped interface **506** also contains a write interface and a

read interface that can support communication with a network processor unit (e.g., NPU 1).

A clock generator circuit **530** and reset logic circuit **532** are also provided. The clock generator circuit **530** may include a delay and/or phase locked loop circuit that is configured to generate internal clock signals that are synchronized with an external clock signal EXTCLK. The reset logic circuit **532** may be configured to perform reset operations when the device **500** is initially powered up or after a chip reset event has occurred. An SRAM interface **534** may also be provided to enable transfer of data to and from an external memory device (e.g., associated SRAM). A cascade interface **536** is provided to support depth-cascading between the search engine device **500**, operating as a "master" device, and a plurality of additional "slave" search engine devices that may be coupled together as illustrated and described more fully hereinbelow with respect to FIG. 8. Other cascading arrangements are also possible.

First and second context sensitive logic circuits **510** and **512** are coupled to the first and second memory mapped interfaces **504** and **506**, respectively. These context sensitive logic circuits **510** and **512** are illustrated as including instruction FIFOs and results mailboxes. The context sensitive logic circuits **510** and **512** may also includes results status circuits that are configured to generate respective aggregate result status signals (ARS) and interrupts, as described more fully hereinabove with respect to FIGS. 3-4. The interrupts may also be used to signify when the age reporting functions may be commenced.

An instruction loading and execution logic circuit **524** is provided with an instruction scheduler **527** and a search and learn (SNL) cache **525**. This logic circuit **524** may perform the functions of a finite state machine (FSM) that controls access to the CAM core **522** and utilizes resources provided by specialized function registers **514**, global mask registers **516**, parity generation and checking circuitry **520** and an aging control logic circuit **518**. The SNL cache **525** may support the performance of search

and learn operations within the CAM core **522**. During search operations, the instruction loading and execution logic circuit **524** provides the CAM core **522** with search words that may be derived from search keys received at a memory mapped interface. In response to a search operation, the CAM core **522** may generate a plurality of hit signals that are encoded to identify an address of a highest priority matching entry within the CAM core **522**. This address may also be encoded as an absolute index that specifies the location of the highest priority matching entry with a multi-chip search machine. In some embodiments, the address may be provided to an index translation logic circuit **526** (ITL). This index translation logic circuit **526** may modify the addresses relative to a selected database to thereby create database relative indexes. Alternatively, the addresses may be modified relative to an NPU-attached associated SRAM to thereby create memory pointer indexes. A results logic circuit **528** is also provided. The results logic circuit **528** is configured to pass results values from the index translation logic circuit **526**, the instruction loading and execution logic circuit **524** and the cascade interface **536** to results mailboxes associated with the context sensitive logic circuits **510** and **512** and the interface logic circuit **508**.

The aging control logic circuit **518** is illustrated as including a plurality of memory devices, which may be updated as each entry is written into the CAM core **522** and during periodic aging operations. These memory devices include a quad arrangement of SRAM memory arrays **700a - 700d**, as illustrated more fully by FIG. 7A. These memory arrays include an age enable memory array **700a**, an age activity memory array **700b**, an age report enable memory array **700c** and an age FIFO select memory array **700d**. In the illustrated embodiment, each bit position within each memory array maps to a corresponding entry within the CAM core **522**. Thus, memory arrays having a capacity of 8k rows and 32 columns will support a CAM core **522** having 256k entries therein. FIG. 7B illustrates in detail how each bit within the age report enable array **700c**

maps to a respective entry within the CAM core **255** having 256k entries (i.e., 262,144 entries).

The data within the age enable memory array **700a** identifies which CAM core entries are subject to aging. For example, each bit position within the age enable memory array **700a** that is set to a logic 1 value (or logic 0 value) may reflect a corresponding CAM core entry that is subject to (or not subject to) aging. Each bit position within the age activity memory array **700b** may reflect whether a corresponding CAM core entry has remained active since the time it was first written into the CAM core **522**. For example, a logic value of 1 may reflect an active CAM core entry that has been the subject of a "hit" during a search operation (or one that has been relatively recently written to the CAM core) and a logic value of 0 may reflect an inactive CAM core entry that is ready to be aged out of the CAM core **522**. Some of the automated aging operations associated with the age enable and age activity memory arrays **700a - 700b** are described more fully hereinabove with reference to FIG. 2B and the age enable and age activity memory arrays **322** and **324** in FIG. 2A.

The age report enable memory array **700c** reflects which entries are to be reported to a command host in response to being aged out of the CAM core **522**. In the event a report only aging feature is provided on a global (i.e., full CAM core), per database and/or per entry basis, the age report enable memory array **700c** may also identify those entries that have exceeded an activity-based aging threshold but have not undergone a final aging out operation (i.e., their valid bits have not been reset to an invalid condition). Thus, a bit position having a logic value of 1 within the age report enable memory array **700c** may identify a corresponding CAM core entry as being subject to age reporting. In contrast, a bit position having a logic value of 0 within the age report enable memory array **700c** may identify a corresponding CAM core entry as not being subject to age reporting when the entry is aged out of the CAM core **522**.

The age FIFO select memory array **700d** reflects where an entry, which is already the subject of age reporting, is reported to upon being aged out of the CAM core **522**. By using one bit per CAM entry, one of two possible age reporting locations is possible. These two age reporting locations include a first FIFO (FIFO 0) and a second FIFO (FIFO 1), which are located within the aging control logic circuit **518**. These FIFOs may each have a capacity of 255 entries. By using a larger memory array, which supports two or more bits per CAM entry, a greater number of age reporting locations may be identified by the age FIFO select memory array **700d**. These first and second FIFOs may be accessed from any of the illustrated interfaces.

The instruction loading and execution logic circuit **524** also operates to control the periodic reporting of the addresses/indexes of the entries from the reporting locations (i.e., FIFO 0 and FIFO 1) to a command host. The phrase "periodic reporting" includes regularly spaced or intermittent reporting that is initiated by the command host or possibly initiated by the IIPC. These reporting operations are performed with the assistance of a plurality of the specialized function registers **514**. These registers **514** include a first level count register and a second level count register. The first level count register is configured to maintain a count of unreported addresses that are stored in aging FIFO 0 and the second level count register is configured to maintain a count of unreported addresses that are stored in aging FIFO 1. The registers **514** also includes a first level configuration register and a second level configuration register. The first level configuration register is configured to maintain a programmable threshold count value that specifies how many addresses can be stored in aging FIFO 0 before the control circuit issues an interrupt to the command host (e.g., NPU 0) to thereby prompt the command host to issue a read request for the addresses stored within aging FIFO 0. Similarly, the second level configuration register is configured to maintain a programmable threshold count value that specifies how many addresses can be stored in

aging FIFO 1 before the control circuit issues an interrupt to the command host (e.g., NPU 1) to thereby prompt the command host to issue a read request for the addresses stored within aging FIFO 1. The registers **514** may also include a first interrupt timer register that operates as a timer to support generation of an interrupt to the command host when no new addresses have been reported to aging FIFO 0 during a programmed time interval and at least one unreported address is stored within aging FIFO 0. This first interrupt timer is used so that the command host (e.g., NPU 0) is aware of the presence of at least one address within aging FIFO 0, even though the threshold count value stored in the first level configuration register has not been exceeded. A second interrupt timer register is also provided to operate in a similar manner with respect to aging FIFO 1.

Aging operations performed by the control circuit of FIG. 5, which includes the instruction loading and execution logic circuit **524** and the aging control logic circuit **518**, include the operations illustrated by FIG. 6. In FIG. 6A, the aging feature of the search engine device **500** may be activated to support age reporting on a per entry basis, Block **600**. Once activated, multiple operations are performed in parallel to generate global aging operation requests and age service requests on a per database basis. At Block **602**, a check is made to determine whether a global aging operation has been requested. If so, a round-robin arbitration operation is performed on any pending database age servicing requests **604**. As illustrated by FIG. 2B, global aging operation requests and database age servicing requests may be generated by programmable aging registers that are configured as countdown counters. The aging counters for those databases that have been programmed to not support aging may be disabled. At Block **606**, an aging operation that supports reporting is performed on an entry within a selected database and then control is returned back to Block **602** to await the next global aging operation request.

Blocks **610 - 616** illustrate a sequence of operations that may be performed to generate each aging operation request on a global basis

within the search engine device. At Block **610**, a countdown operation is commenced in a global aging register and a check is continuously made at Block **612** to determine whether a countdown operation has completed. If so, an aging operation is requested (see, Block **602**) and the global aging register count is reloaded into the global aging register, Block **616**.

Blocks **618** - **624** illustrate operations that may be used to generate age service requests for respective databases. If a CAM core is configured to support a maximum of 16 databases, then sixteen sets of operations corresponding to Blocks **618** - **624** are performed in parallel at potentially different frequencies. As illustrated by Block **618**, a countdown operation is performed on a database aging register at a specified frequency. When the count reaches zero, an age service request is issued for the corresponding database Blocks **620** - **622**. At Block **624**, the corresponding database aging register count is reinitialized and the operations are repeated. The database aging register count values should be sufficiently high to prevent a backlog of age service requests for a given database when the round-robin arbitration of the database age servicing requests is performed, Block **606**.

As illustrated by FIG. 6B, operations **606** for performing aging on a selected entry within a selected database include a checking operation to determine whether a selected entry is subject to aging, Block **632**. This operation includes checking the corresponding bit position within the age enable memory array **700a** to determine whether the entry is subject to aging. If the selected entry is subject to aging, then a check is made to see if the entry is active or not, Block **636**. If the age activity memory array **700b** indicates that the entry is active (e.g., the age activity bit is set to 1), then the corresponding age activity bit is reset and the aging operation is complete, Block **634**. However, if the entry is not active (e.g., the age activity bit is set to 0), then a check is made at Block **637** to determine whether report-only aging is enabled. If report-only aging is enabled, then Block **638** is bypassed. The report-only aging feature may be established

on a global basis (e.g., by setting an AR ONLY GLOBAL bit within an aging control circuit **518**) or per database basis (by setting an AR ONLY bit within a corresponding database configuration register (see, e.g., registers **514**). When the report-only aging feature is applied to an entry that is scheduled to be aged out (i.e., Block **636** decision results in a "NO" conclusion, which means the entry has exceeded an activity-based aging threshold), an address of the entry may be reported to an aging FIFO, but the entry will not be aged out by having its validity bit reset.

If report-only aging is not enabled, then the selected entry is removed from its database (e.g., the entry is marked as invalid using a CLEAR VALID instruction that causes an access to the CAM core **522**), Block **638**. An entry may be marked as invalid by resetting the validity bit for the entry. Alternatively, a predetermined data string having a validity bit that is set to an invalid state may be written over the aged out entry. This may be particularly helpful in those embodiments that support background error detection and/or correction with parity and/or Hamming code bits. In some cases, the value of the validity bit may influence the value of the parity and/or Hamming code bits and merely resetting the validity bit when performing an age out operation may cause the entry to be improperly detected as invalid (and then possibly corrected by setting the validity bit to a valid state) during a background error detection and/or correction operation. To prevent the unintentional correction of an aged out entry, the predetermined data string having correct parity and/or Hamming code bits may be used as a default word that is to be written over every entry that is to be aged out of the CAM core.

As illustrated by Block **639**, the corresponding age enable bit within the age enable memory array **700a** is cleared so that the selected entry is no longer evaluated for aging (see, Block **632**). A check is then made to determine whether the selected entry is subject to reporting to the command host (e.g., NPU 0, NPU 1 or PCI), Block **640**. This check can be performed by evaluating the corresponding bit position within the age report

enable memory array **700c**. Accordingly, even if a selected entry is identified at Block **637** as being subject to report-only aging at a global or per database level, the check at Block **640** may override these settings for a given entry.

If the aged entry is subject to reporting, then the age reporting enable setting for the entry is cleared, Block **641**, and the address/index of the entry is added (i.e., "reported") to either FIFO 0 or FIFO 1, Block **642**. The destination FIFO to which the aged entry is added is controlled by the value of the corresponding bit position within the age FIFO select memory array **700d**. If the aged entry is reported to FIFO 0, then the identity of the aged out entry will ultimately be read from one of the memory mapped interfaces. Alternatively, if the aged entry is reported to FIFO 1, then the identity of the aged entry will ultimately be read from another one of the memory mapped interfaces. The timing of these read operations is a function of the timing of when the respective command hosts (e.g., NPU 0, NPU 1 or PCI), which issue the FIFO read instructions, receive corresponding interrupts that identify FIFO 0 or FIFO 1 as being sufficiently full. In the event FIFO 0 or FIFO 1 becomes completely full before being emptied by a command host, the instruction loading and execution logic **524** may operate to suspend age reporting or even operate to suspend all aging operations until such time as the age reporting FIFOs have been emptied.

The control circuit within the search engine device **500** may also be configured to fill FIFO 0 and FIFO 1 with the addresses of entries that have been aged out of other search engine devices. For example, when the illustrated search engine device **500** is configured as a master search engine device within a depth-cascaded search machine, the cascade interface **536** will operate to pass the indexes of aged out entries from one or more "slave" search engine devices to the aging FIFOs within the master search engine device. Accordingly, as illustrated by FIG. 8, a multi-chip search machine **800** may include a cascaded age reporting path that

operates to pass the addresses/indexes of aged out entries along the cascaded chain of slave search engine devices (shown as NSE 1 - NSE 7) to the cascade interface of the master search engine device (shown as NSE 0).

Referring now to FIG. 9, a CAM-based search engine device **900** according to further embodiments of the present invention operates to prevent the learning of duplicate entries within a database when the learning operations are performed in response to search and learn (SNL) instructions issued by a command host. In FIG. 9, an instruction loading and execution logic circuit **524** is illustrated. Aspects of this instruction loading and execution logic circuit **524** were previously described hereinabove with respect to FIG. 5. This logic circuit **524** may receive instructions (and supporting data) from a plurality of instruction FIFOs, shown as IF0, IF1 and IF2. These instruction FIFOs may constitute the instruction FIFOs illustrated in Blocks **508**, **510** and **512** of FIG. 5. The logic circuit **524** may generate instructions to the CAM core **522** and receive results (e.g., hit or miss signals) from an output of the results logic circuit **528**.

The logic circuit **524** is illustrated as receiving a plurality of instructions. According to one environmental example, these instructions may include a search instruction (with Search Key 0) from IF2, a write instruction (with Search Key 1) from IF1, and two equivalent SNL instructions (with Search Key 2) from IF0 that are pipelined into the search engine device **900** in consecutive sequence. In alternative examples, these two equivalent SNL instructions may be received from different instruction FIFOs and be associated with different contexts. The logic circuit **524** arbitrates to determine the sequence of handling the competing instructions and access to the CAM core **522**. As described herein, SNL instructions are deemed equivalent when they are associated with same search keys and directed at the same database(s) within the CAM core **522**.

The handling of the two equivalent SNL instructions by the logic circuit **524** and CAM core **522** of FIG. 9 will now be described more fully with respect to the flow diagram of FIG. 10. This discussion assumes that the two equivalent SNL instructions from IF0 in FIG. 9 are scheduled as two immediately consecutive instructions and the other two instructions from IF1 and IF2 are scheduled before or after the equivalent SNL instructions are processed. In FIG. 11, the other two instructions are illustrated as being scheduled after the equivalent SNL instructions.

As illustrated by FIG. 10, a sequence of operations **1000** associated with a search and learn instruction may include the issuance of an SNL instruction by a command host (shown as an NPU), Block **1002**. This SNL instruction may be received by an instruction FIFO and then passed to the instruction loading and execution logic circuit **524**, which includes a scheduler **527**, Block **1004**. The finite state machine within the logic circuit **524** identifies the instruction as an SNL instruction, Block **1006**. As illustrated by Block **1008**, a search instruction (i.e., search portion of the SNL instruction) and associated search key are transferred to the CAM core **522**. In response to this transfer, a search operation is performed on a selected database(s) within the CAM core **522**. This search operation will result in a hit or miss condition. A check is made at Block **1020** to determine whether the search operation resulted in a hit condition or not. Concurrently with the transfer of the search instruction and search key to the CAM core **522**, an operation is performed to add the search key to the SNL cache memory device **525** within the logic circuit **524**, Block **1010**. This SNL cache memory device **525** may operate as a first-in first-out (FIFO) memory device having a predetermined capacity (e.g., 32 entries). In particular, the capacity of the SNL cache memory device **525** should be sufficient to support the operations described herein even under worst case latency conditions. These worst case latency conditions may occur when a depth-cascaded chain of search engine devices are provided and the corresponding database to which an SNL instruction applies is located in

the last search engine device within the chain. Under these conditions, the SNL cache memory device **525** in the master search engine device (see, e.g., NSE 0 in FIG. 8), which may be used to keep track of all SNL instructions applied to all search engine devices within the cascaded chain, needs to have sufficient capacity to prevent a duplicate learn operation from occurring in the corresponding database even when a pair of equivalent SNL instructions that are directed to that database are spaced apart from each other in time by a substantial number of clock cycles.

The operation to add a new search key to the SNL cache memory device **525** may constitute a "push" operation onto a FIFO memory "stack." An operation is then performed to determine whether the newly added search key is a duplicate of a search key currently residing in the SNL cache memory device **525**, Block **1012**. If a duplicate search key is not present, then the search key is marked with a learn instruction, Blocks **1014** and **1016**. However, if a duplicate search key is present, then the search key is marked with a search instruction instead of a learn instruction, Blocks **1014** and **1018**. These marking operations may cause the generation of opposite flag values associated with each entry in the FIFO memory device (e.g., flag=1 means the search key is marked with a search instruction and flag=0 means the search key is marked with a learn instruction). These flag values may constitute "marker" information.

Returning to Block **1020**, if the search portion of the SNL instruction results in a hit condition, then this hit condition and a corresponding index of a matching entry are returned to a results logic circuit (see, Block **528** in FIG. 5). The corresponding search key within the SNL cache memory device **525** is also removed, Block **1022**, and operations associated with the SNL instruction terminate. However, if the search portion of the SNL instruction results in a miss condition, then a check is made at Block **1024** to determine whether the corresponding search key in the SNL cache memory device **525** is marked as a duplicate. If the search key is not marked as a duplicate, then the CAM core **522** undergoes a learn operation

with the search key and the search key is removed from the SNL cache memory device **525** (i.e., "popped" from the FIFO memory stack), Block **1028**. This learn operation may result in a writing of the search key into a next free address within a specified database in the CAM core **522** and a return of a result indicating the index of the CAM row that received the search key, Block **1030**. Operations associated with the performance of a learn instruction are more fully described in commonly assigned U.S. Application Serial Nos. 10/620,161, filed July 15, 2003, and 10/688,353, filed October 17, 2003, the disclosures of which are hereby incorporated herein by reference. Alternatively, if the check at Block **1024** indicates that the search key has been marked as a duplicate, then a search operation using the search key is performed on the CAM core **522**, Block **1026**, and the search key is removed from the SNL cache memory device **525**. As described more fully hereinbelow with respect to FIGS. 11A-11H, this second search operation associated with a single SNL instruction should result in a hit condition and return the index of a matching entry within the specified database, Block **1030**.

The operations illustrated by FIG. 10 will now be described more fully to illustrate how two equivalent SNL instructions are handled within the search engine devices **500** and **900** when they are processed under "worst case" timing conditions that are most likely to result in a duplicate learn operation if conventional SNL instruction handling operations are used (i.e., as immediately consecutive instructions). The timing conditions that typically cause duplicate learn events in a CAM core may vary as a function of instruction latency. For example, if the latency between the generation of an instruction (e.g., LEARN) to the CAM core and the return of a corresponding result from the CAM core is sufficiently long, then many different timing conditions may result in duplicate learn events. In particular, as the latency of processing through a CAM core (or multiple CAM devices within a cascaded chain) increases, the number of cycles that may be spaced between two equivalent learn instructions that are likely to

cause a duplicate learn event also typically increases. Accordingly, even timing conditions that do not represent worst case timing conditions (i.e., immediately consecutive learn instructions) may contribute to duplicate learn events in conventional search engine devices.

In FIG. 10, the first of these two SNL instructions will be designated as SNL_1 and the second of these two SNL instructions will be designed as SNL_2. The timing of these instructions assumes that no prior equivalent SNL instructions have been received by the logic circuit **524** and the search key is not already present as a valid entry within the CAM core **522**.

The first SNL instruction SNL_1 and search key (Search Key 2) are transferred to the CAM core **522** and the search key is transferred to the SNL cache **525**, Blocks **1008** and **1010**. A search of the SNL cache **525** is then performed to detect the presence of a duplicate search key. This search of the cache results in a miss, Block **1014**. As illustrated by Block **1016**, the search key is marked with a learn instruction, which means a flag may be set that designates the search key as one that is to accompany a learn instruction when it is subsequently read from the SNL cache **525**. At Block **1020**, a check is made to determine whether a search of the CAM core **522** resulted in a hit or miss. Because the CAM core **522** did not contain the search key (i.e., Search Key 2), the check will result in a miss result. Then, at Block **1024**, the flag associated with the search key in the SNL cache **525** will be checked to see whether it designates an attached learn instruction (key is not marked as a duplicate) or whether it designates an attached search instruction (key is marked as a duplicate). Because the search key is marked with a learn instruction, the search key and learn instruction are transferred to the CAM core **522** and the search key (Search Key 2) is learned, Block **1028**. Thus, the first SNL instruction results in a search operation followed by a learn operation. In response, the CAM core **522** is updated with a new entry (Search Key 2).

At possibly the same time as the first search operation of SNL_1 is being checked at Block **1020**, the second SNL instruction SNL_2 and

search key (Search Key 2) are transferred to the CAM core **522** and the search key is transferred to the SNL cache **525**, Blocks **1008** and **1010**. At Blocks **1012** and **1014**, the search key will be marked as a duplicate because the earlier equivalent search key is still held by the SNL cache **525**. This means a flag may be set that designates the search key as one that is to accompany a search instruction when it is subsequently read from the SNL cache **525**, Block **1018**.

At Block **1020**, a check is made to determine whether a search of the CAM core **522** resulted in a hit or miss. Because the CAM core **522** has not yet learned the search key as a result of the SNL_1 instruction, this check will result in another miss result. Then, at Block **1024**, the flag associated with the search key in the SNL cache **525** will be checked to see whether it designates an attached learn instruction (key is not marked as a duplicate) or whether it designates an attached search instruction (key is marked as a duplicate). Because the search key is marked with a search instruction, the duplicate search key and search instruction are transferred to the CAM core **522** and the search operation is performed, Block **1026**. At Block **1030**, the results of this second search operation associated with SNL_2 are processed. These results include an indication of a hit condition (because of the earlier learn operation associated with SNL_1) and an index of the matching entry. Accordingly, rather than having two SNL instructions result in duplicate learning events into a database (because they arrive too close in time for the first SNL instruction to take effect before the search portion of the second SNL instruction is performed), the second SNL instruction is converted into a search and search (SNS) instruction, which results in a hit condition and returns an address of the learned entry back to a results mailbox.

This sequence of operations is also illustrated by FIGS. 11A-11H. At FIG. 11A, two equivalent SNL instructions (SNL_1 and SNL_2) are illustrated as being received by the scheduler **527** within the instruction loading and execution logic circuit **524**. These instructions are followed by

a write instruction (Key 1) and a search instruction (Key 0), which may be handled in a conventional manner. At FIG. 11B, the SNL cache **525** is illustrated as including the new search key (Search Key 2) and a flag indicating that the search key is associated with a learn instruction. This key and flag are illustrated by the reference KEY2(L). Although not shown in FIG. 11B, the SNL cache **525** is preferably designed to retain additional information along with the search key and flag. This additional information may include a database identifier and other related information. The CAM core **522** is also illustrated as commencing a first search operation with the search key, in response to SNL_1.

At FIG. 11C, the SNL cache **525** is illustrated as including a duplicate search key and a flag indicating that the duplicate search key is associated with a search instruction. This duplicate key and flag are illustrated by the reference KEY2(S). The CAM core **522** is also illustrated as commencing a second search operation with the search key, in response to SNL_2. At FIG. 11D, the first search operation associated with SNL_1 is illustrated as resulting in a miss result, which is passed back to the logic circuit **524**. A write operation using Key 1 is also illustrated. In FIG. 11E, the search operation associated with SNL_2 is illustrated as resulting in a miss result and a learn instruction with the search key is added to the scheduler **527**. This learn instruction constitutes the learn portion of SNL_1. A search operation using Key 0 is also illustrated.

In FIG. 11F, the learn instruction and search key are passed to the CAM core **522** and the second search instruction associated with SNL_2 is added to the scheduler **527**. Here, the learn portion of SNL_2 is converted into a search instruction in order to prevent a duplicate learning event. The search instruction with Key 0 is also illustrated as resulting in a miss condition. In FIG. 11G, the learn instruction is illustrated as generating a learned address result (i.e., address of Search Key 2 within the CAM core **522** is returned to results mailbox and then passed back to a command host). Finally, in FIG. 11H, the search instruction associated with SNL_2 is

illustrated as generating a hit address result (which reflects the fact that SNL_1 resulted in a correct learn of a new entry and SNL_2 resulted in a correct search based on the newly learned entry instead of a duplicate learn of the search key and a negative hit result).

One potential limitation associated with the above-identified operations has to do with the processing of equivalent SNL instructions when a corresponding database to which the SNL instructions apply is full. In such a case, the first SNL instruction will not result in a successful learn operation and the marking of duplicate entries within the SNL cache **525** may result in repeated searches of the CAM core **522** and possibly an absence of learn instructions to update the CAM core **522** when the corresponding database is finally free to accept a new entry. To avoid this potential limitation, operations may be performed to clear one or more duplicate flag settings associated with the related SNL cache entries when a corresponding database (to which the search key is to be learned) is full. In particular, a configuration register associated with the registers **514** (see, FIG. 5) may retain a "SNL_Clear_All_Duplicates" bit that identifies whether one (bit = 0) or all (bit = 1) of a plurality of related duplicate flag settings with the SNL cache **525** will be cleared whenever the corresponding database is full. Clearing one or all of the duplicate flag settings will enable a duplicate SNL instruction to retain its learn component operation and thereby update the corresponding database within the CAM core **522** when the database gains free entries.

Referring now to FIG. 12, operations **1200** that illustrate additional methods of processing instructions according to embodiments of the present invention will be described. These operations, which may be performed by Blocks **524**, **522** and **528** in FIG. 5, assume that the SNL cache **525** of FIG. 5 has been replaced by a searchable instruction cache (I_CACHE). This cache may be configured as a content addressable memory (CAM) instruction buffer that supports varying search key widths. This CAM-based I_CACHE may be subject to periodic aging operations to

removed unwanted entries (e.g., old entries). The frequency of the aging operations should be sufficient to prevent the I_CACHE from becoming full. At Block **1202**, a check is made to determine when an incoming instruction received by the instruction loading and execution logic **524** is a SEARCH instruction. If not, a check is then made to determine whether the incoming instruction is a LEARN instruction, Block **1204**. If the incoming instruction is neither a SEARCH instruction nor a LEARN instruction, the instruction is inserted into the instruction pipeline, Block **1224**, and then passed to the CAM core, Block **1226**.

However, if the incoming instruction is a LEARN instruction, then a search is made of the I_CACHE to detect the presence of an equivalent search key (i.e., same key value and same database identifier), Block **1206b**. At Block **1208b**, a check is made to determine whether an equivalent search key was detected based on the search at Block **1206b**. If an equivalent search key is not present, then the search key is added as an entry to the I_CACHE and a duplicate bit associated with the search key entry is set (e.g., duplicate bit is set to 1 binary), Block **1214**. The instruction insertion operations starting at Block **1224** are then performed. But, if an equivalent search key is present based on the check at Block **1208b**, then a check is made of the I_CACHE to determine whether a duplicate bit for the search key has been asserted, Block **1212**. If not, then the duplicate bit is set (i.e., asserted) at Block **1216** and control is passed to Block **1224**. If yes, the LEARN instruction is blocked, Block **1222**, and control is passed to Block **1224**, where the CAM core may experience of no-op cycle. Although the learn instruction is blocked, additional operations may be performed to update a results mailbox to indicate that the search key associated with the blocked instruction was previously learned.

Referring again to Block **1202**, if a SEARCH instruction is detected, then control is passed to Block **1206a**, where a search of the I_CACHE is performed to detect an equivalent search key. If an equivalent search key is not present, Block **1208a**, then control is passed to Block **1224**. But, if an

equivalent search key is present, then a check is made to determine whether the corresponding duplicate bit is asserted, Block **1210**. If a duplicate bit is asserted, then control is passed to Block **1224**. If the duplicate bit is not asserted, then the duplicate bit is set, Block **1218**, and the SEARCH instruction is converted into a LEARN instruction, Block **1220**, before control is passed to Block **1224**.

Once an instruction has been inserted into the instruction pipeline at Block **1224**, the instruction (e.g., SEARCH, LEARN, WRITE, READ, etc.) is performed within the CAM core, Block **1226**. If the result of a CAM core operation indicates that a search has been performed and a MISS result has been generated, Block **1228**, then the corresponding search key is added to the I_CACHE, Block **1230**, and control is passed to Block **1232** where results of a CAM core access are processed. (See, e.g., FIGS. 3-4).

The operations illustrated by FIG. 12 will now be described more fully using multiple examples of instruction sequences that illustrate how the presence of an I_CACHE within the instruction loading and execution logic **524** can operate to at least reduce the occurrence of unintentional duplicate learn events within a CAM core. In a first example, two equivalent LEARN instructions, which have the same key and are directed to the same database, are scheduled for insertion into the instruction pipeline as two immediately consecutive instructions. This timing represents a worst case scenario where a duplicate learn event is most likely to occur using conventional instruction processing operations. These two LEARN instructions may be issued by the same command host or by different command hosts that are supporting different contexts within the search engine device **500**. The first LEARN instruction passes to Block **1206b** where a search of the I_CACHE is made to determine whether the equivalent search key is stored therein. Assuming that an equivalent search key is not already present, then the search key (and database identifier) are stored within the I_CACHE and the corresponding duplicate bit is set, Block **1214**. The instruction is then inserted into the instruction

pipeline and passed to the CAM core as a LEARN instruction, Blocks **1224** and **1226**. This LEARN instruction will cause the CAM core to be updated with the new search key in the designated database. The address of the newly learned entry will then be returned to a corresponding results mailbox (for the given context) and thereafter communicated to the command host that issued the corresponding LEARN instruction.

The second LEARN instruction also passes to Block **1208b** where its search key is compared with the entries in the I_CACHE. Because of the earlier I_CACHE update caused by the first LEARN instruction, the check at Block **1208b** results in an affirmative answer. A check to determine whether the corresponding duplicate bit has been asserted is then performed, Block **1212**. This check also results in an affirmative answer (based on the earlier learn of the equivalent search key) and control is passed to Block **1222**. At Block **1222**, the second LEARN instruction is blocked in order to prevent a a duplicate learn event from occurring within the CAM core.

In a second example, two equivalent SEARCH instructions, which have the same key and are directed to the same database, are scheduled for insertion into the instruction pipeline as two spaced apart instructions. This example assumes the database does not contain the search key. At Blocks **1202** and **1206a**, a check is initially performed to determine whether the first instruction is a SEARCH instruction and then a search is made of the I_CACHE to detect the presence of an equivalent search key. For purposes of this example, this search of the I_CACHE results in a negative result, Block **1208a**, and control is passed to Block **1224**. At Blocks **1224** and **1226**, a first SEARCH operation is performed on the CAM core. A MISS result is returned in response to the first SEARCH operation and the I_CACHE is updated with the corresponding search key, Blocks **1228** and **1230**. The MISS result in then processed, Block **1232**.

Assuming now that the lag time associated the second SEARCH instruction relative to the first SEARCH instruction enables the I_CACHE to

be updated before the second SEARCH instruction is inserted into the pipeline, then the second SEARCH instruction results in a search of the I_CACHE, which is performed at Block **1206a**. The result of this search indicates the presence of the equivalent search key, Block **1208a**. Then, at Block **1210**, a check is made to determine whether the duplicate bit associated with the equivalent search key is asserted. Because the duplicate bit has not been set, control is passed to Blocks **1218** and **1220**. At Block **1218** the duplicate bit is set and at Block **1220** the second SEARCH instruction is converted into a LEARN instruction. This LEARN instruction is inserted into the instruction pipeline, Block **1224**, and then the operations illustrated by Blocks **1226**, **1228** and **1232** are performed. At Block **1232**, the address of the entry that received the new search key during the LEARN operation is passed to a corresponding results mailbox and the command host is ultimately notified of the entry address corresponding to the second SEARCH instruction. In this manner, the I_CACHE may be used to not only prevent duplicate learn events, as described in the first example, but may also be used in certain circumstances to block repeated MISS results from occurring in response to repeated equivalent search operations. If this feature is not necessary, then the instruction loading and execution logic **524** may be programmed so that the operation illustrated by Block **1202** is not performed and the operations illustrated by Blocks **1206a**, **1208a**, **1210**, **1218** and **1220** are bypassed.

In the drawings and specification, there have been disclosed typical preferred embodiments of the invention and, although specific terms are employed, they are used in a generic and descriptive sense only and not for purposes of limitation, the scope of the invention being set forth in the following claims.